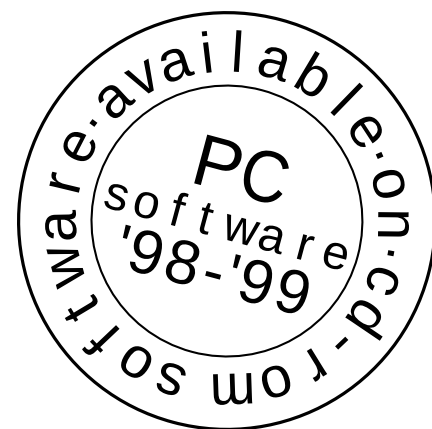


The winning design among the Competition entries received by our Dutch sister magazine is a MicroPascal compiler for microcontrollers from Intel's MC51 series. The software enables these microcontrollers to be programmed in a simple way using the 'Pascal' higher programming language. Helped by a separate ROM emulator, MicroPascal may considerably reduce software debugging time.

Software by J. van de Kamer

# MicroPascal

## First Prize, Netherlands



ables, etc.) is then gathered and stored together with the identifier. The examination of the program continues until an error is encountered, or a full stop (.) is found behind a command. In the compiler, every command is processed as a procedure. If necessary, such a procedure may call itself (this is called 'nesting').

During the compilation process, the first-time appearance of a new command causes the associated machine code to be generated. Because the compiler has a flexible structure, the commands to be generated are stored in a special library file called MCS51.DLL.

Before the actual compiler operation starts, the requisite library has to be loaded into the memory.

Initially the compiler does not know the addresses, interrupts and labels to be used. Consequently, it will first generate so-called relocatable (address-independent) code. This code is stored as a file with the extension 'MPU' (for MicroPascal Unit). This intermediate code must not be changed by the user because the compiler will assume that it has been generated without errors. The file format is universal, allowing the MPU file to be used in combination with other controllers, too.

During the second phase, the file is 'linked', which means that labels, procedures and functions set up in the source code file are coupled to real addresses. The result is a file that may be programmed into an EPROM. For this purpose, MicroPascal supports two file formats: Intel-hex and CPULink.

As the author is ready to admit, designing a complete compiler involves a vast amount of complex work. Mr. van de Kamer started off within a Borland Pascal environment, but eventually changed to Delphi 1.0, while the final touches to the compiler were made in Delphi 3.0. After about two years of programming activity (approximately 750 hours) and an impressive 29,731 lines of source code, the compiler was ready in the form as submitted for the Software Design Contest launched in the July/August 1998 issue of *Elektor Electronics*.

### Compiling is translating

The function of a compiler may be compared to that of a translator. Commands from a higher program-

ming language are converted (translated) into commands from a lower-level language. During the compiling process, the translator module will typically encounter three types of 'word':

- reserved words such as the commands Begin and End;
- words representing a value, for example, '1' or 'Joe';
- words used as a label.

Comment enclosed in braces {} is ignored during the compile process. The syntax of a source code file is checked using a fixed procedure. The first word that has to be found is Program or Init. If not, an error report is immediately returned. Once the right header is found, the translator will start to look for the next one, in this case, an identifier. All relevant information on the program (name, constants, vari-

## Memory use

The RAM memory is employed for the storage of variables, procedures, a software ALU (arithmetic Logic Unit), a stack and intermediate results (scratchpad)

Memory is filled from the top to the bottom. For the internal RAM, the top is \$FF, for the external RAM, \$7FFF (provided 32 Kbytes of external RAM is used). Global variables are held in this external memory. One word, for example, occupies address \$7FFF for the 'high' byte and \$7FFE for the 'low' byte. A string made up of 10 bytes is stored in 11 bytes: 10 bytes for the characters and number 11 to indicate the length. The last character is stored at location \$7FFF, the first at \$7FF6, and the length indicator at \$7FF5.

## Software ALU

To be able to perform arithmetic operations using variables with a size larger than one byte, a software ALU is built into the program. The size of the ALU is geared to the largest value supported by the program. The ALU is divided into two equal sections, lo\_ALU and hi\_ALU. The ALU is an important piece in the memory. It is used to move variables to and from the stack, and perform arithmetic operations.

## Other points of interest

Once procedures and functions are being used, local variables are created that only exist within the relevant procedure or function. Obviously, the compiler has to reserve space for these intermediate variables. The simplest approach is to define memory space for any procedure which pops up in the program. In practice, this results in a lot of wasted memory space. The present compiler first looks for the procedure or function which may be expected to use up the largest number of variables (in bytes). For this purpose, a special buffer area is made available behind the software ALU. All local variables generated by the program are stored in this area. The remainder of the RAM memory is available as stack space. MicroPascal works 'top down', starting at the highest memory location (\$FF). This system had to be adopted because the controller uses memory locations from \$00 upwards for its internal registers. The processor's own stack is then located immediately above these registers. Both blocks are therefore allowed to grow towards each other as the program is being executed, and no con-

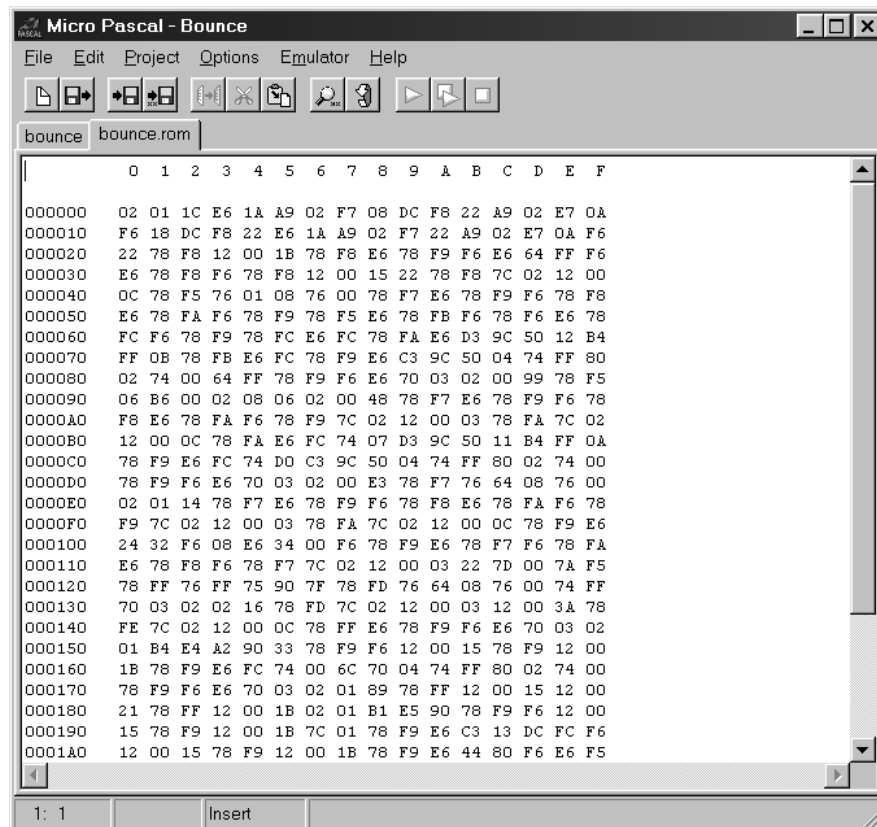


Figure 1. Screenshot of MicroPascal in action. The compiler offers a simple way of developing code for microcontrollers from the MCS-51 family. The screens in this article show the source code in a window (introductory illustration), and the final machine code ready for programming into a PROM or EPROM.

flict will occur as long as sufficient memory space remains available. If you run low on available memory, simply reduce the number of variables and/or procedures being called. When calling a procedure it is also possible to convey parameters or variables. If variables are marked with 'VAR' they may be adapted within the procedure called. To be able to use interrupts, special provisions have to be made in the code. This is necessary because an

interrupt may occur at any moment. The interrupt procedure ensures that all data of local variables, as well as ALU data, are safely stored before an interrupt procedure is started. The internal registers are also kept in a safe place. At the end of the interrupt, all relevant information is retrieved and restored to its original state. Interrupts may be used to adapt globally defined variables. We should hasten to add, however, that variables larger than one byte can not be

| Variables and units                                                                                                                |           |              |                |
|------------------------------------------------------------------------------------------------------------------------------------|-----------|--------------|----------------|
| Type                                                                                                                               | value     | minimum      | maximum        |
| Byte                                                                                                                               | numerical | 0            | 255            |
| Word                                                                                                                               | numerical | 0            | 65535          |
| Shortint                                                                                                                           | numerical | -128         | 127            |
| Integer                                                                                                                            | numerical | -32768       | 32767          |
| Boolean                                                                                                                            | numerical | False        | True           |
| Char                                                                                                                               | character | 1 character  | 1 character    |
| String                                                                                                                             | character | 0 characters | size dependent |
| For a string, the size depends on allocated memory space.<br>When no length is specified, the length is limited to 255 characters. |           |              |                |

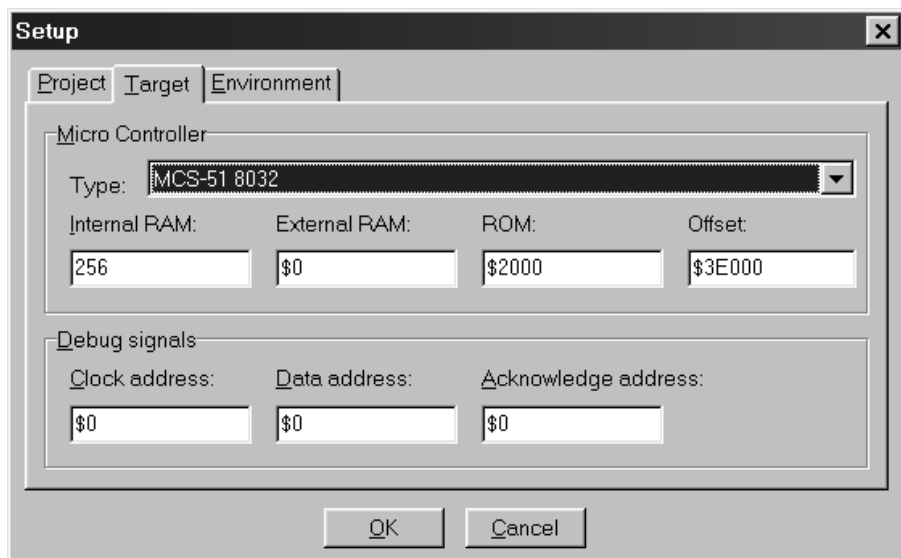


Figure 2. A special configuration window allows the characteristics of the MCS-51 system ('target system') to be set up.

processed. Larger variables are modified with the aid of several instructions having a width of one byte. To communicate with the main program, it is therefore recommended to use variables with a width of one byte.

## Debugging

Eliminating errors in software is almost invariably a tedious and time-consuming activity. MicroPascal has a number of extra functions available to simply the debugging process. Additional hardware has been designed in the

form of a ROM emulator (size 256 Kbytes). The circuit diagram and PCB artwork for this design may be found on the CD-ROM which contains a number of prize-winning entries from the 1998 Software Design Competition. This CD-ROM will be published early January 1999. The relevant file is called ROM emulator.DOC.

During the linking of a program, each debugging location has its own identification (a value between 0 and 255). This value is saved by the compiler, together with the location of the

debugging point (file name and line number). The debugging procedure is implemented in the software by calling a function from the library that first sends the identification, and then the contents of the local and global memory. MicroPascal 'knows' which variables are visible at the debugging point marked by the identification, and displays the received value plus associated variables.

This approach requires three I/O pins of the MCS-51 to be given a specific function. Two outputs, one carrying the clock signal, the TxD line, plus one input employed for handshaking. A simple handshaking protocol is employed. The controller pulls bit 0 at the data output logic low, and places a low level on the clock line. MicroPascal reads the level and pulls the acknowledge line logic low. Next, the controller pulls the clock signal high again, whereupon MicroPascal does the same with the acknowledge signal. The first bit is then transmitted. This procedure is completed for all bits that have to be sent, except the last one. With this bit, MicroPascal no longer supplies an acknowledge signal, causing the microcontroller to enter an infinite loop. During this period, the results may be viewed and analysed on the PC display. Once all relevant information is known, you only have to actuate the menu option 'Continue' to transmit one acknowledge signal and so get the microcontroller out of its infinite loop. The program is then continues as before.

The primitive handshake was chosen to make sure reliable communication is available under all circumstances. However, the price of three controller I/O pins may be too high in some cases, so that the function is optional.

## System requirements

MicroPascal runs under Windows 95 and requires about 10 Mbytes of hard disk space. If the optional ROM emulator is employed, then you also need a free bidirectional printer port.

MicroPascal was developed for MCS-51 microcontrollers and comes with a library for these processors only. Users requiring support for other microcontrollers will have to develop their own libraries.

(992012-1)

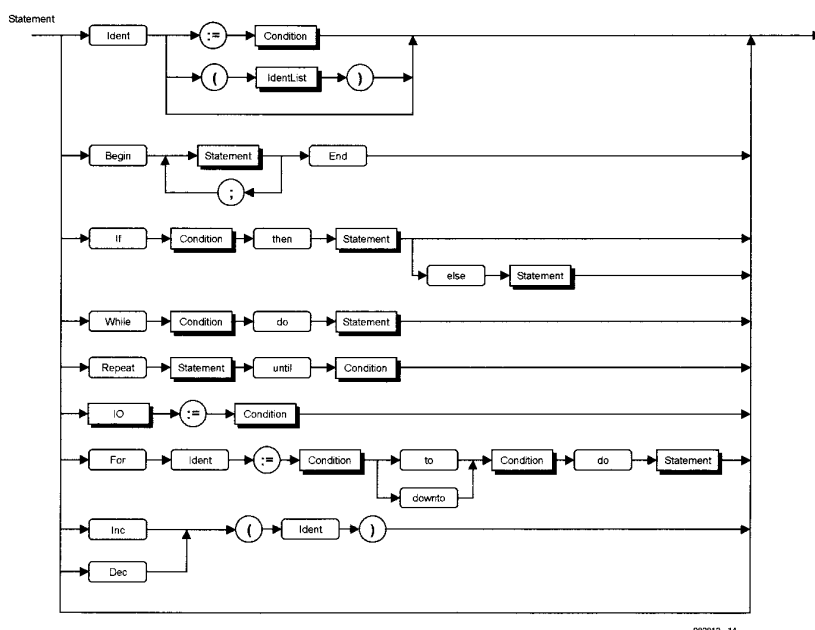


Figure 3. This flow diagram shows the ingenious methods used by the compiler to determine what is meant by a statement in the source code.